21.4 Polynomial time reductions
21.4.1
A quick review of polynomial time reductions
**Polynomial-time reductions**

We say that an algorithm is **efficient** if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem $X$ to problem $Y$ (we write $X \leq_P Y$), and a poly-time algorithm $A_Y$ for $Y$, we have a polynomial-time/efficient algorithm for $X$. 

![Diagram](image)
Polynomial-time reductions

We say that an algorithm is **efficient** if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem $X$ to problem $Y$ (we write $X \leq_P Y$), and a poly-time algorithm $A_Y$ for $Y$, we have a polynomial-time/efficient algorithm for $X$.

![Diagram of polynomial-time reductions](image)
Polynomial-time reductions

We say that an algorithm is **efficient** if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem $X$ to problem $Y$ (we write $X \leq_p Y$), and a poly-time algorithm $A_Y$ for $Y$, we have a polynomial-time/efficient algorithm for $X$. 

![Diagram of polynomial-time reductions](image)
Polynomial-time reductions

We say that an algorithm is **efficient** if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem $X$ to problem $Y$ (we write $X \leq_p Y$), and a poly-time algorithm $A_Y$ for $Y$, we have a polynomial-time/efficient algorithm for $X$.
Polynomial-time Reduction

A polynomial time reduction from a decision problem \(X\) to a decision problem \(Y\) is an algorithm \(A\) that has the following properties:

1. given an instance \(I_X\) of \(X\), \(A\) produces an instance \(I_Y\) of \(Y\)
2. \(A\) runs in time polynomial in \(|I_X|\).
3. Answer to \(I_X\) YES \(\iff\) answer to \(I_Y\) is YES.

Proposition 21.1.

If \(X \leq_p Y\) then a polynomial time algorithm for \(Y\) implies a polynomial time algorithm for \(X\).

Such a reduction is called a Karp reduction. Most reductions we will need are Karp reductions. Karp reductions are the same as mapping reductions when specialized to polynomial time for the reduction step.
Polynomial-time Reduction

A polynomial time reduction from a decision problem $X$ to a decision problem $Y$ is an algorithm $A$ that has the following properties:

1. given an instance $I_X$ of $X$, $A$ produces an instance $I_Y$ of $Y$
2. $A$ runs in time polynomial in $|I_X|$.
3. Answer to $I_X \text{ YES} \iff \text{answer to } I_Y \text{ is YES.}$

Proposition 21.1.

If $X \leq_P Y$ then a polynomial time algorithm for $Y$ implies a polynomial time algorithm for $X$.

Such a reduction is called a Karp reduction. Most reductions we will need are Karp reductions. Karp reductions are the same as mapping reductions when specialized to polynomial time for the reduction step.
Review question: Reductions again...

Let $X$ and $Y$ be two decision problems, such that $X$ can be solved in polynomial time, and $X \leq_p Y$. Then

(A) $Y$ can be solved in polynomial time.
(B) $Y$ can NOT be solved in polynomial time.
(C) If $Y$ is hard then $X$ is also hard.
(D) None of the above.
(E) All of the above.
THE END
...
(for now)